システムLSI 検証技術入門

デバイスの記事

第14回

論理回路のプロパティ検証技術（6）

――アサーション・ベース検証の基礎

藤田昌宏

ここでは，アサーション・ベース検証について解説する．アサーション・ベース検証は，検証対象となる設計の内部に観測点を設け，この点に注目して設計バグを検出する手法である．バグの信号値を外部端子に導いて確認する従来手法では検出しにくいケースでも，比較的容易にバグを検出できるという．学会などでは，この手法を用いることにより，検証期間が約1/2になったと報告されている．（編集部）

今回は，アサーション記述言語を実際の設計に利用する際の考えかたについて説明したいと思います．これは，一般に「アサーション・ベース検証」と呼ばれています．シミュレーション，形式的検証のいずれにも利用でき，実際の設計にも取り入れられ始めています．

● チップ・レベル検証への依存度が高い

図1に，現在，広く利用されているLSI設計とその検証の流れを示します．仕様は，設計対象のチップ全体として与えられ，設計作業ではその仕様を詳細化していきます．その際に，チップ全体を徐々に分割していき，分割されたものに対する仕様を定義していくことになります．チップ・レベルでは，メモリ，CPU，IP（intellectual property）コア，そして外部とのインターフェースという単位で分割されます．この過程では，設計対象のシステムLSIに対するハードウェア部分とソフトウェア部分の分割なども実施されます．

CPUについては，既存のIPコアがすでに用意されており，新たに設計する必要はありません．設計対象となるLSIの中のカスタム・ハードウェアの部分が詳細化・分割の対

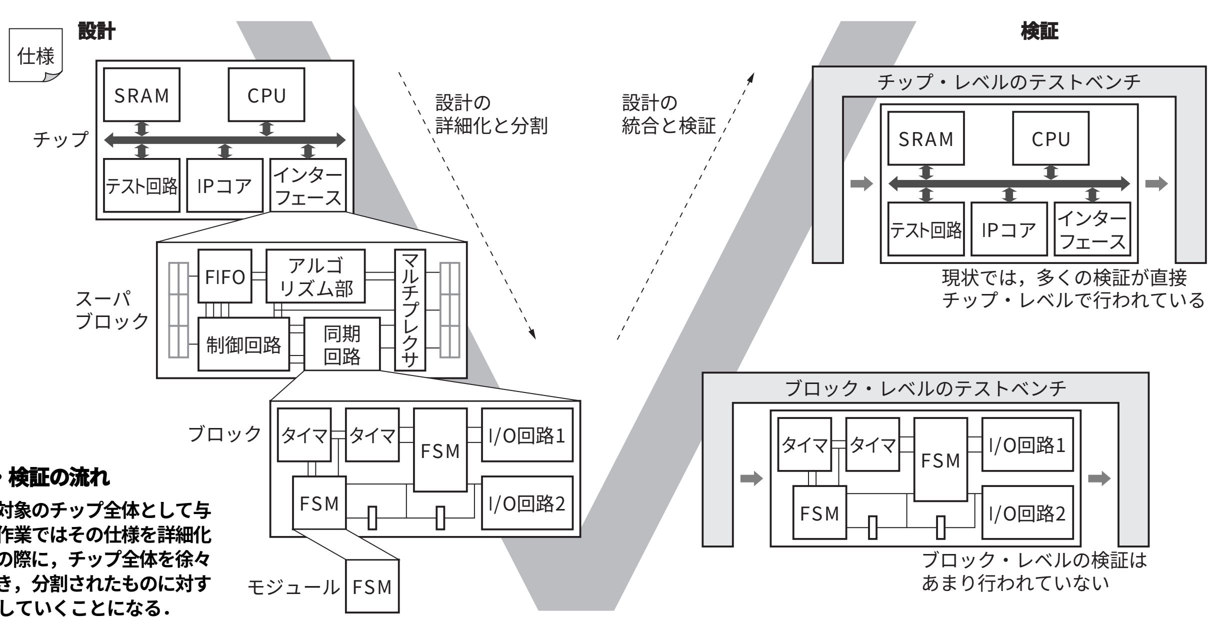


図1
現状の設計・検証の流れ
仕様は，設計対象のチップ全体として与えられ，設計作業ではその仕様を詳細化していく．その際に，チップ全体を徐々に分割していき，分割されたものに対する仕様を定義していくことになる．

象となります．**図1**の例ではインターフェースがカスタム・ハードウェアであり，詳細化していきます．この部分は，FIFO（first-in first-out）などのバッファ・メモリ，なんらかの処理を行うアルゴリズム部，また，外部との通信の同期をとる部分，そして，外部ポートのマルチプレクサや制御回路などに分割できます．この比較的大きなブロックのことを，ここではかりにスーパブロックと呼ぶことにします．

さて，このスーパブロックはさらに詳細化され，ブロックの集まりに分解されます．**図1**の例では，通信の同期をとる部分について詳細化しています．これは，制御を実現する状態遷移（FSM：finite state machine），I/O回路，そしてタイマなどから構成されます．さらに設計を詳細化し，いわゆるモジュール単位に分割してRTL（register transfer level）記述を作成することになります．

以上のように設計の各部分がモジュール・レベルまで詳細化されると，次にそれらを結合しながら，検証することになります．ほんとうは設計の詳細化の各段階ごとに検証を行っていくことが望ましいのですが，設計期間に制限があったり，各モジュール，ブロック，スーパブロックに対する仕様がかならずしも明確になっていない場合も多く，全部をつないで行うチップ・レベル検証への依存度が高くなっているようです．

● ブロック・レベルでバグを検出することが理想だが…

本連載でも説明してきたように，形式的検証が利用できる局面は，現在のところかなり限られます．チップ全体を扱える形式的検証ツールは，現状では組み合わせ回路の等価性検証用だけであり，LSIの設計データを製造部門に引き渡す際に，その設計データともとのRTL記述の間の等価性をチェックする程度です．

しかし見かたを変えると，設計規模さえある水準以下であれば，形式的検証ツールが利用可能であるとも言えます．**図1**の検証の流れでは，モジュールやブロックといった小規模回路については十分に利用可能なはずです．アサーション・ベース検証とは，このモジュールやブロックの検証に対して，とくに形式的検証（実際には，アサーション・ベース検証をシミュレーションによって実施することも多い）をうまく利用できるようにする技術であると言えます．

検証をチップ・レベルで行う場合，まず問題となるのが，チップ全体の設計データがそろわなければ作業が始められないという点です．つまり，設計作業がある程度進んでからでないと，検証を始められないことになってしまいます．

また，設計規模の点から形式的検証手法が適用できず，シミュレーションを行うことになるのですが，一般に，チップ全体に対するシミュレーションだけでは検証カバレッジ（網羅率）を高くすることが困難です．検証カバレッジでは，例えばRTL記述にある条件文に対して，条件が成立する場合と成立しない場合の両方をチェックしないとカバレッジが上がりませんが，チップの外部端子からのみ信号パターンを供給していると，なかなか内部の状態を思うように制御できません．結果として，きわめて低いカバレッジしか得られないか，あるいは膨大な入力パターンが必要となってしまいます．また，後で例を示しますが，設計バグのため，内部の動作におかしな点があっても，その影響が外部端子に伝わるような入力パターンになっていないと，バグを外部から観測できません．

さらに，テストベンチの再利用も難しくなってしまいます．チップ全体に対するテストベンチであるため，まったく同じ機能のチップでないとそれを再利用できません．現在，システムLSIの全開発期間に占める検証期間の割合が増大しており，テストベンチなど，検証データの再利用が課題となっていますが，それもたいへん難しいわけです．

これに対して，ブロック・レベルで検証すれば，設計作業のより早い段階で検証を始められます．また，検証対象規模がそれほど大きくない場合も多く，形式的検証ツールも利用しやすくなります．ここでブロック・レベル検証とは，チップ全体ではなく，その中の各ブロックを切り出してきて，それを検証対象とすることを指します．

しかし，ブロック・レベル検証がそれほど簡単に実施できないのも事実です．まず，各ブロック単位でテストベンチを用意する必要があるわけですが，それはかならずしも容易ではありません．各設計者は，自分が担当するブロックの動作はよく理解していますが，もし，検証を行う人が別の人だった場合，各ブロックの動作や仕様がかならずしも明確ではなく，テストベンチの作成にかなりの時間がかかってしまいます．この理由として，各ブロック間のインターフェースがかならずしも明確に定義されていない，ということが挙げられます．

これはよくないことなのですが，設計しながら各ブロック間のインターフェース仕様を決めていったり，各設計者が自分で適当に決めているといった場合も少なくありません．結果として，ブロック・レベルでもテストベンチの再

利用が難しいということが問題となります．しかし，ブロック・レベル検証が不十分だと，チップ・レベル検証に，より多くの期間がかかってしまいます．

● アサーションでブロック・レベル検証を強化

以上の問題を解決することを目指して考えられたのがアサーション・ベース検証と呼ばれる技術です．これは，各ブロック内にアサーションと呼ばれる「設計が満たさなくてはならないこと」，あるいは「設計が満たしてはならないこと」の条件を設計記述の間に挿入し，シミュレータや形式的検証ツールでそれをチェックするという技術です．

アサーション・ベース検証という考えかたそのものは決して新しいものではありません．ソフトウェア開発では，ポインタがどこも指していないこと（null pointer）などをチェックするコードをプログラムに挿入する手法が利用されてきています．これは，アサーション・ベース検証の考えかたそのものです．また，ハードウェア設計者も，設計の動作を監視するモニタと呼ばれるモジュールを設計記述に埋め込んだり，設計中のエラーの条件を調べて，それが成り立った場合にメッセージを表示する（例えば，Verilog HDLなら，`if (err == 1) $display ( ... );` ）などして，アサーションを利用しています．さらに，論理合成などで使われる各種のディレクティブも，一種のアサーションであるといえます．

アサーション・ベース検証とは，これら従来から設計者が行ってきたことを共通のアサーション言語と検証環境で実現できるようにし，結果としてテストベンチなどの再利用を図りやすくした技術であると言えます．すなわち，アサーションを利用して設計者の意図を設計記述に埋め込むことであり，シミュレータなどの検証ツールから見ると，それらの埋め込まれたものが「実行可能なコメント」に見えるというものです．

● アサーションと制約で可観測性，可制御性を改善

アサーション・ベース検証の基本的な考えかたを**図2**に示します．アサーション・ベース検証では，設計の中の信号値が特定の値であることをチェックするアサーションと，信号値に対する制限を表現する制約（constraints）を設計記述の中に挿入し，アサーションによって内部の信号値の可観測性（observability）を，制約によって可制御性（controllability）を向上させ，検証期間の短縮を図る技術です．**図2**はチップ・レベルの検証の例ですが，もちろんブロック・レベルでも同様の方法を利用できます．

また，アサーションによって複数ブロック間のインターフェース条件も記述できます．**図3**を見てください．ブロックAとブロックBが接続されている状況で，ブロックBの検証を行いたいとします．このとき，ブロックAとやりとりされる信号の値はブロックAの動作に依存するわけですが，ブロックAもいっしょに検証しようとすると検証対象規模が増大してしまいます．そこで，ブロックAの抽象的なふるまいをアサーションの形で表現し，ブロックBとこの

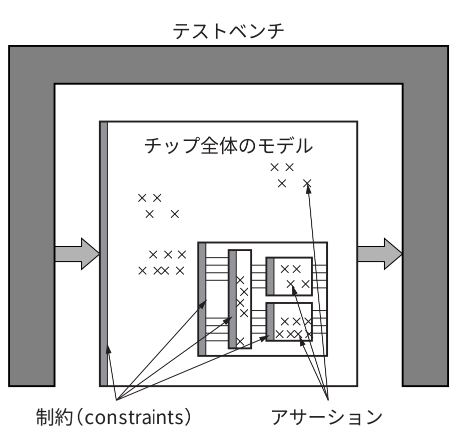


図2　アサーション・ベース検証の考えかた

設計の中の信号値が特定の値であることをチェックするアサーションと，信号値に対する制限を表現する制約（constraints）を設計記述の中に挿入する．アサーションによって内部の信号値の可観測性（observability）を，制約によって可制御性（controllability）を向上させる．この手法はチップ・レベル検証だけでなく，ブロック・レベル検証にも適用できる．

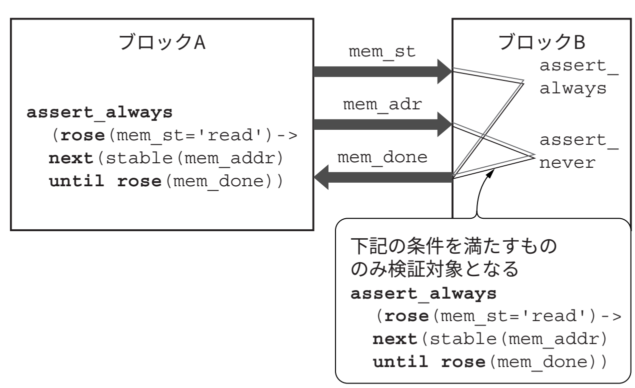


図3　インターフェースに関するアサーション

ここではブロックBの検証を行うことを考える．ブロックAとやりとりされる信号の値はブロックAの動作に依存するが，ブロックAもいっしょに検証しようとすると検証対象規模が増大してしまう．そこで，ブロックAの抽象的なふるまいをアサーションの形で表現し，ブロックBとこのアサーションを利用して検証するようにすれば，検証対象規模の増大を防ぐことができる．このようにすると，ブロックAからの信号はアサーションを満たすものしか許されないので，ブロックBの内部の検証もそれに沿った信号値のみを対象に行われることになり，検証効率が上がる．

アサーションを利用して検証するようにすれば，検証対象規模の増大を防ぐことができます．このようにすると，ブロックAからの信号はアサーションを満たすものしか許されないので，ブロックBの内部の検証もそれに沿った信号値のみを対象に行われることになり，検証効率が上がります．

なお，このようなアサーションは形式的検証だけでなく，シミュレーションでも利用できます．このブロックAのアサーションのことをインターフェース・アサーションと呼びます．インターフェース・アサーションで制限された信号値に対してのみ検証を行うわけです．

以上をまとめると，アサーションを利用することにより以下の利点があります．

- 内部の信号値に対する可制御性と可観測性を改善できる
- 設計の早期の段階から，各ブロックに対する検証を開始できる（チップ全体のブロックを接続する前に検証できる）
- ブロック単位に適用したアサーションを再利用できる可能性がある
- アサーションそのものは，形式的検証でもシミュレーションでも同じように利用できる

● バグの値を設計内部のアサーションで検出

では，少し例を見てみましょう．ここでは**図4(a)**のような回路を検証することを考えます．部分回路が三つあり，いくつかのフリップフロップ（FF）で接続されています．そして，いちばん上のFFの入力信号に設計バグがあったとします．この場合，検証とは設計バグがあることを認識し，その場所を特定することです．一般に設計バグの存在を知るためには，そのバグがある部分の信号の値を出力端子に伝播させる必要があります．

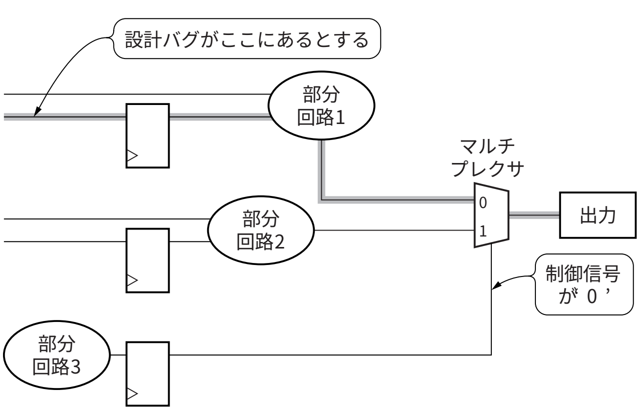
図4(a)において，出力端子につながっているマルチプレクサの制御信号を‘0’にできると，設計バグの値を出力まで伝播させることができます．しかし，マルチプレクサの制御信号は部分回路3によって制御されています．部分回路3の出力値が‘1’になってしまうと出力端子まで設計バグの値が伝播せず，結果として設計バグを発見できないことになります．

部分回路3がどれくらい複雑かにもよりますが，例えばランダム・シミュレーションでは，部分回路3の出力がなかなか‘0’にならないということも十分に考えられます．また，部分回路3が別の回路で制御されている場合，出力値を制御することがさらに難しくなります．場合によっては，人手でシミュレーション・パターンを作成することすら困難なケースも考えられます．

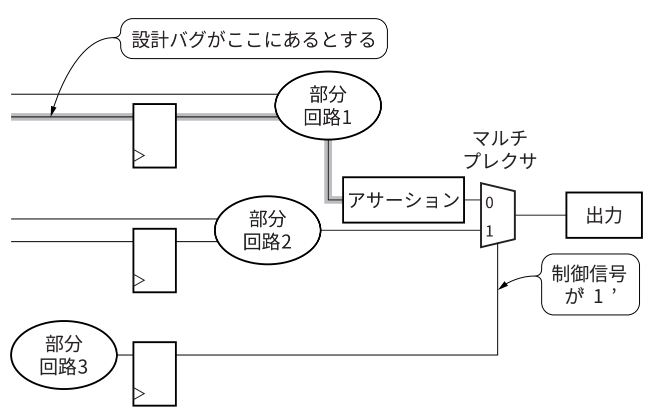
そこで，出力端子まで設計バグの値を伝播させるのではなく，設計の中の適切なところにアサーションをうまく挿入してやると，設計バグを比較的容易に検出できます．例えば**図4(b)**では，マルチプレクサの入力側にアサーションを挿入してやると，簡単に設計バグを発見できることになります．

● アサーション記述は再利用しやすい

アサーション・ベース検証において重要なことの一つと



(a) 出力端子でバグを検出できる場合



(b) アサーションを挿入してバグを検出

図4 アサーション・ベース検証の効果

(a)のように，出力端子につながっているマルチプレクサの制御信号を‘0’にできると，設計バグの値を出力まで伝播させることができる．しかし，マルチプレクサの制御信号は部分回路3によって制御されており，部分回路3の出力値が‘1’になってしまうと出力端子では設計バグを検出できなくなる．例えば，(b)のようにマルチプレクサの入力側にアサーションを挿入してやると，簡単に設計バグを発見できる．

して，1組のアサーション記述（設計対象のアサーションの集合）をさまざまな用途で利用できることがあります（**図5**）．

まず，利用するアサーションの作成ですが，自分で設計対象に依存するアサーションをいろいろと試行錯誤しながら作成します．その際に，以前の設計と同じ仕様となっている部分については，以前の設計に対するアサーションをライブラリからうまく抽出すれば，再利用できます．また，設計対象が標準的な仕様に基づいている場合（例えば，利用するバスがAMBAバスなど，広く利用されているバス規格になっている場合），アサーションを検証用IPとして外部から調達できる場合があります．

このように，さまざまな方法でアサーションを集めて，設計対象に対する十分なアサーションの集合を構成することができます．しかし，逆に言うと，十分なアサーションを集められなければ，以後の検証も不十分になる可能性が高く，いかに十分なアサーションを集められるかが重要な課題となります．その意味では，同種の設計に対するアサーションを集め，ライブラリ化して保守していくという手法がきわめて重要になります．

次にアサーションの利用ですが，**図5**に示すように，シミュレーションによる検証，および形式的検証ツールのプロパティとしての利用が考えられます．また，人手などでテストベンチを作成する際にも参考になります．さらに，シミュレーションで検証カバレッジを計算する場合に，アサーションを満たす範囲でのみカバレッジを計算するなどの利用方法も考えられます．前途のインターフェース・アサーションなどは，このような方法で利用されます．

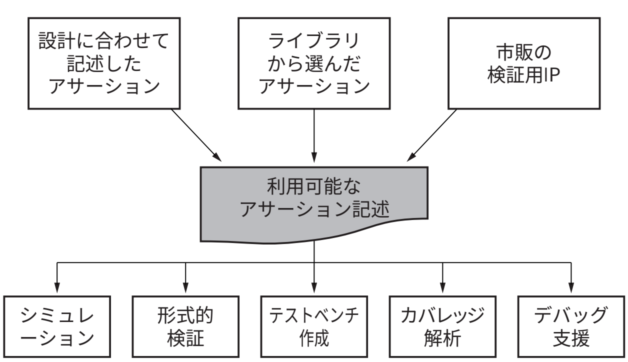


図5　アサーションは各種ツールで利用できる

アサーション記述は，設計に合わせて一から作成したものだけでなく，過去に作成してライブラリに入っていたものや，市販の検証用IPとして提供されているものも利用できる．用意したアサーション記述は，シミュレーションや形式的検証のほか，テストベンチの作成やカバレッジ解析，デバッグなどにも流用できる．

シミュレーションや形式的検証によって設計バグが発見された場合，具体的にどのアサーションが不成立なのかを調べていくと，設計の内部のどの部分が意図したとおりに動作していないのかを調べられます．アサーションを使わずにデバッグを行ったときより，内部の調査が容易になります．

● アサーションを使えば検証期間が約1/2に

では，実際，アサーション・ベース検証にはどのくらいの効果があるのでしょうか？ 効果の例を大ざっぱに示した図が**図6**です．大ざっぱな例と言ったのは，実際にこのような数値となる設計事例があったわけではなく，いくつかの設計経験を総合すると，だいたいこのような感じの数値になることが多いという意味です．

このグラフの横軸は，検証のために費やしたコンピュータの稼働時間（CPU時間）で，縦軸は，その結果発見した設計バグ数の合計を表しています．実際のところ，設計の中にいくつのバグがあるかは不明ですが，検証を続けてもバグを発見できなくなった時点で検証がほぼ完了したと考えるのが普通です．グラフの下の曲線はアサーション・ベース検証ではない通常の手法を利用した際のもので，上の曲線はアサーション・ベース検証を利用した場合のものです．

通常の検証手法を利用した場合，20万CPU時間の検証を行っても，発見できたバグの数は310個にとどまっています．これに対して，アサーション・ベース検証では5万CPU時間の時点でほぼ同等の300個の設計バグを発見できており，20万CPU時間では400個の設計バグが発見されています．したがって，このあとどれだけ検証を続けるかにもよりますが，アサーション・ベース検証を行うことで，

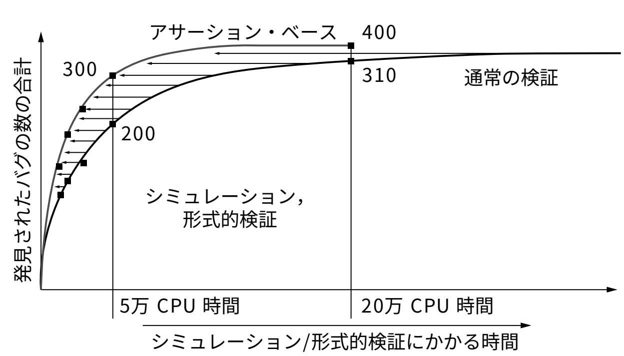


図6　アサーション・ベース検証の期待される効果

実際にこのような数値となる設計があったわけではない．いくつかの設計経験を総合すると，だいたいこのような感じの数値になることが多かった．

検証期間を半分程度にできるという発表が行われています．例えば，各種の学会発表では次のようなことが報告されています．

- DEC Alpha 21164の設計では，34%の設計バグをアサーション・ベース検証で速やかに検出できた
- DEC Alpha 21264の設計では，25%の設計バグをアサーション・ベース検証で速やかに検出できた
- Cyrix M3（p1）の設計では，17%の設計バグをアサーション・ベース検証で速やかに検出できた
- Cyrix M3（p2）の設計では，50%の設計バグをアサーション・ベース検証で速やかに検出できた
- 米国Hewlett-Packard社の発表では，85%程度のバグをアサーション・ベース検証で検出できた
- 米国Intel社の発表では，アサーションを形式的検証に利用することで，400個程度の設計バグを発見できた

● 標準バスのアサーション記述は検証用IPとして流通

上でも少し述べたように，共通に使われている仕様，例えば標準バス・プロトコルに関するアサーション記述は汎用的に利用できます．そのため，検証用IPとして提供されているものも少なくありません．例えば，PCIバス，AMBA（Advanced Microcontroller Bus Architecture）バス，USB，DDR（double data rate）SDRAMなどのメモリ・バスについて検証用IPが検証ツール・ベンダなどから提供されています．これらを購入することにより，容易に網羅的なアサーション・ベース検証を行えます（**図7**）．

ただし，どこのベンダの検証用IPも内容が同じかというと，そうとは限りません．同じAMBAバス・プロトコル用であっても，各社がおもに自然言語で記述されたAMBAバス・プロトコルの仕様を個別に解釈してアサーションを作成しているので，アサーション記述は異なるはずです．その意味では，信頼できる検証用IPかどうかを判断することも，重要な課題となってくると考えられます．

実際の設計現場では開発期間が限られることから，アサーションを書いている時間があれば，設計のほかの作業にその時間を使いたいと考える設計者も多いと思います．現状で検証期間が大幅に増大していることを考えると，開発期間の中でアサーションを記述するための時間をきちんと確保しておくことも必要になるでしょう．また，アサーションを書いていると，アサーションのデバッグ，つまり，意図したとおりのアサーションになるように適宜修正するための時間が必要になり，結果として検証期間が短くならないと考える設計者もいるかもしれません．

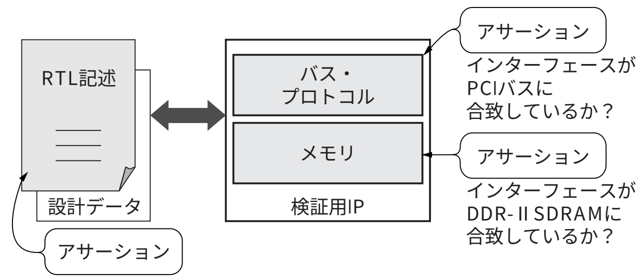


図7　検証用IPの考えかた
標準的に使われているバス・プロトコルについては，検証用IPとして提供されているものが少なくない．例えば，PCIバス，AMBAバス，USB，DDR SDRAMなどのメモリ・バスについて，検証用IPが検証ツール・ベンダなどから提供されている．

しかし，既存のテストベンチ作成でも同じようなデバッグが必要だったわけであり，この点は従来と変わらないと思います．また，各種の学会報告によると，アサーションのデバッグに時間がかかったとしても，合計の検証期間は短くなっています．アサーション記述を挿入すると，シミュレーション時間が10～30%程度延びることを気にする設計者もいますが，これもアサーション・ベース検証による検証期間短縮の効果を考えると問題とはなりません．

さて，今回は，アサーション・ベース検証について説明しました．これでプロパティ検証に関する話は終わりにして，次回からは，少し先端的な検証のトピックをいくつか説明したいと考えています．例えば，今いろいろと話題になっているビヘイビア記述（高位記述）とRTL記述の等価性検証の具体例を紹介していきたいと思います．

ふじた・まさひろ
東京大学 大規模集積システム設計教育研究センター（VDEC）教授

<筆者プロフィール>
藤田昌宏．1985年，東京大学大学院情報工学専門課程修了．工学博士．同年，富士通研究所入社．論理設計用CAD技術の研究開発に従事．1993年より米国富士通研究所へ出向，CAD研究グループを立ち上げる．2000年3月より東京大学大学院工学系研究科電子工学専攻教授，2004年4月より東京大学大規模集積システム設計教育センター（VDEC）教授．ディジタル・システム設計支援技術，特にシステム・レベル合成，論理合成，論理検証を中心に研究している．